**Spark installation**

* Can be used in anaconda, colab, databricks(online platform), or locally install
* To setup in anaconda – create environment in anaconda using below command

**conda create -n pyspark-env python=3.7**

* Activate it

**conda activate pyspark-env**

* Install pySpark

**conda install -c conda-forge pyspark**

* Launch jupyter

**jupyter notebook**

* Create a PySpark Notebook: In Jupyter Notebook, create a new notebook and import PySpark

**from pyspark.sql import SparkSession**

**spark = SparkSession.builder.appName("PySparkExample").getOrCreate()**

Is a bigdata processing framework

Open source, scalable, parallel, in-memory execution environment for running analytics or in-memory layer that sits above multiple data stores where data can be loaded to memory & analysed across the cluster

**Difference between mapreduce and spark**

Mapreduce distributes files across the disk whereas spark works in-memory which makes it faster

**Features of Spark**

**Speed** – swift processing framework – 100times faster in in-memory execution and 10 times faster on disk comparing to Hadoop mapreduce

**Ease of use** – we will be writing programs in Java, Scala, Python, R

**Generality** – Combine SQL, streaming and complex analytics

**Runs everywhere –** runs on Hadoop, Apache Mesos, Kubernetes, standalone or in cloud

**Cache** – has simple programming layer that provides powerful caching and disk persisitence capabilities

**Deployment** – can be deployed through messages or spark’s cluster manager

**Realtime** – it is developed for real-time processing so offers real-time competition and low latency because of in-memory execution

**Pyglot spark** – provides high level apis in java, scala, python and R. provides a shell in scala and python

**Components of spark ecosystem**

**Core** – responsible for basic i/o functions, scheduling, monitoring etc

Entire apache spark is built on top of core execution engine

Has extensible api’s in languages like scala, pyhton , R, Java

**Spark ecosystem library** is composed of various components like spark SQL, spark streaming(batch processing and streaming), machine learning library and GraphX(to work with graphics)

**Programming –** supports Scala, java, python, R

**Storage –** stores data in HDFS, amazon S3, SQL and NoSQL db’s

**RDD(Resilient distributed dataset)**

Distributed computing in Hadoop needs storage of data in HDFS and multiple I/O operations makes it slower, they were replications and serialization which makes even slower.

* Our goal was to reduce num of I/O operations from HDFS, it is made possible by in-memory data sharing faster than disk and network sharing
* Backbone of spark
* Fundamental datastructure/schema structured – handles both structured and unstructured data
* Reads data from rdd, performs transformations on old rdd and creates new 1, performs some actions on rdd and stores that data from rdd to a persistent storage.
* Rdd is immutable, can contain any type of python, scala or java object
* Data set present in rdd is divided into logical partitions which may be computed on different cluster nodes, becos of this transformations and actions can be parallely made
* RDD are resilient meaning they can be easily recovered from any issues, even if 1 executional node fails as it stores data on multiple executional nodes even if 1 fails, we still can process data

**RDD features**

1. **In-memory compution**
2. **Lazy evaluation –** transformations are not calculated until action is applied
3. **Fault tolerant** – tracks the data lineage information & rebuilds the data automatically.
4. **Immutability –** data can be created or retrieved but can’t be changed
5. **Partioning –** fundamental unit of parallelism and all data chunks are divided into partitions
6. **Persistance –** users can reuse rdd
7. **Coarse Grained Operations –** applies to all elements in data sets through maps or filters or group by operations

**RDD creation**

1. From parallelized collections
2. From existing RDD or
3. From external data sources like hdfs, s3,edge base etc